MEASURING ENERGY CONSUMPTION Using python

Project Title: **Measuring Energy Comsuption**

Phase 5: **Project Documentation & Submission**

Problem Definition :

The challenge we aim to address is the creation of an automated system for measuring energy consumption, performing data analysis, and presenting visualizations to facilitate informed decision-making. The objective is to improve efficiency, accuracy, and comprehension in managing energy consumption across diverse sectors.

Design Thinking

1) Data Source:

We will begin by sourcing data from an available dataset containing energy consumption measurements. For this project, we will use the dataset provided at the following link: [Hourly Energy Consumption Dataset](https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption).

2) Data Preprocessing :

Before analysis, we'll perform data preprocessing to ensure data quality and usability. This includes tasks suchas handling missing values, data cleaning, and transforming the dataset into a suitable format for analysis.

3) Feature Extraction :

To extract meaningful insights, we'll identify and extract relevant features and metrics from the energy consumption data. These could include hourly consumption patterns, seasonality, and trends.

4) Model Development :

Statistical analysis and machine learning techniques will be applied to the dataset to uncover patterns, trends, and anomalies in energy consumption. This will involve building predictive models to understand consumption behavior.

5) Visualization :

Effective visualization is crucial for conveying insights. We 'll develop various visualizations such as line charts, bar graphs, and heatmaps to present energy consumption trends, patterns, and anomalies. Visualization tools like Matplotlib and Seaborn will be utilized.

6) Automation :

To streamline the entire process, we will create a script or program that automates data collection, analysis, and visualization. This automation will ensure real-time or periodic updates, making the system efficient and user-friendly.

Project Roadmap:

1. **Data Collection and Preprocessing**
   - Acquire the energy consumption dataset.
   - Perform data cleaning and preprocessing.

2. **Feature Extraction**
   - Identify key features for analysis.
   - Extract relevant metrics from the dataset.

3. **Model Development**
   - Apply statistical analysis and machine learning algorithms.
   - Build predictive models for energy consumption.

4. **Visualization**
   - Create visualizations to present insights.
   - Develop an interactive dashboard for user-friendly access.

5. **Automation**
   - Build a script to automate data collection, analysis , and visualization.
   - Ensure real-time or scheduled updates.

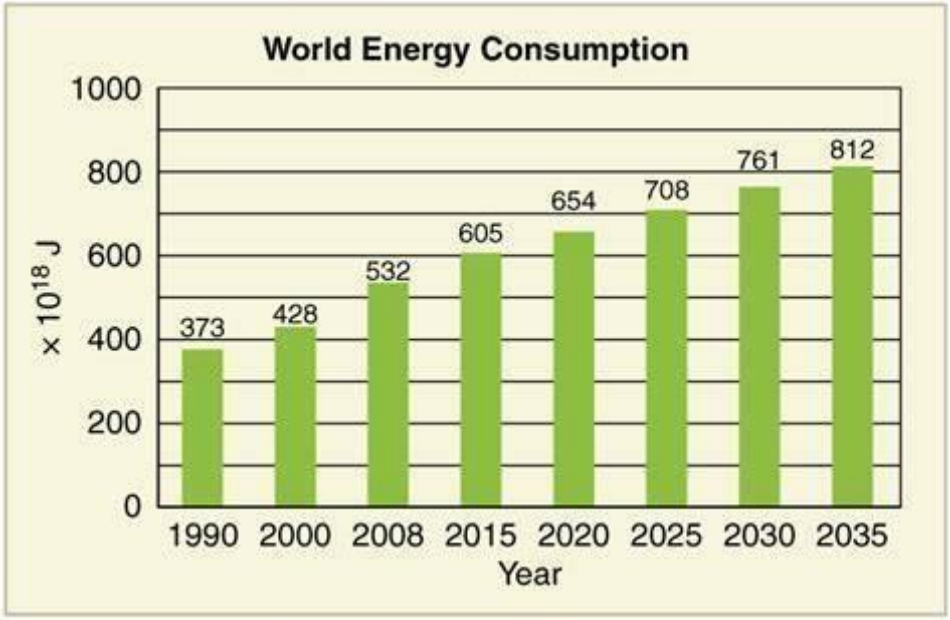
6. **Testing and Validation**
   - Thoroughly test the system for accuracy and reliability.
   - Validate the results against real-world data.

7. **Documentation and Deployment**
   - Prepare documentation for users and developers.
   - Deploy the automated system for practical use.

Time series analysis

SEASONAL DECOMPOSITION: DECOMPOSE HISTORICAL ENERGY CONSUMPTION DATA INTO ITS SEASONAL, TREND, AND RESIDUAL COMPONENTS TO UNDERSTAND RECURRING PATTERNS.

- **EXPONENTIAL SMOOTHING: METHODS LIKE HOLT-WINTERS CAN CAPTURE TRENDS AND SEASONALITY IN TIME SERIES DATA.**



- **AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA): A WIDELY USED MODEL FOR FORECASTING BASED ON HISTORICAL VALUES, DIFFERENCING, AND LAGGED VALUES.**
- **FOURIER ANALYSIS: UTILIZE FOURIER TRANSFORMS TO IDENTIFY PERIODIC COMPONENTS IN ENERGY CONSUMPTION DATA.**

Machine Learning Models:

- Regression Models: Linear regression, polynomial regression, or ridge regression can be applied to capture relationships between energy consumption and various factors like temperature, day of the week, and more.
- Decision Trees and Random Forests: These models can handle non-linear relationships and interactions in the data.
- Neural Networks: Deep learning models, such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), are effective for time series forecasting.
- XGBoost and Gradient Boosting: These ensemble methods excel in capturing complex patterns and are robust against overfitting.

Feature Engineering:

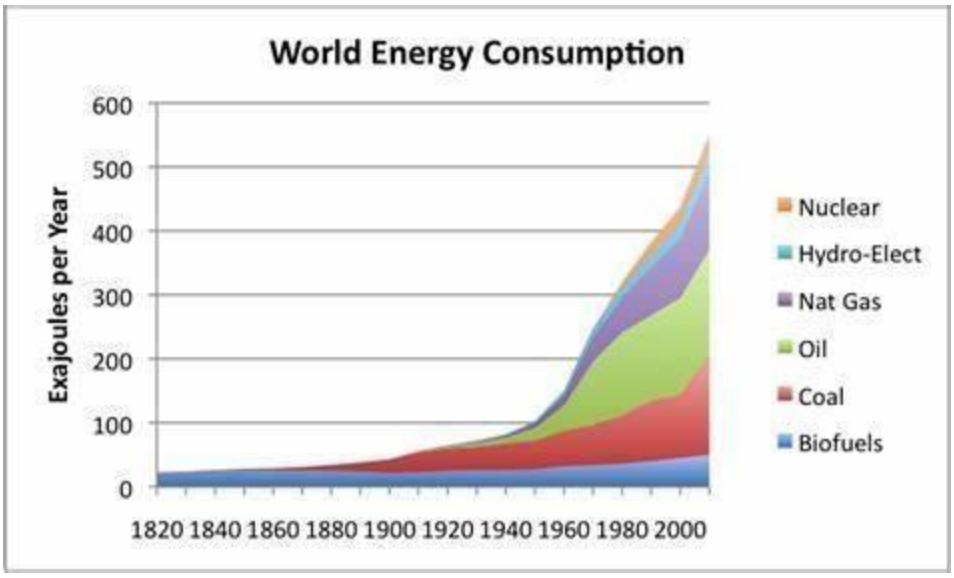
- Create relevant features like time of day, weather conditions, holidays, and economic indicators to improve model accuracy
- In this section, the aforementioned feature engineering methods are applied to an illustrative dataset. Descriptions of this dataset are given in Section 3.1;
- the feature importance information with different feature engineering methods are discussed in Section 3.2;
- finally visualization of feature space is provided in Section 3.3 to give a better understanding about how the proposed feature engineering methods work

Evaluation and Validation:

-
- Use metrics like Mean Absolute Error (MAE), Root Mean Square Error (RMSE), or Mean Absolute Percentage Error (MAPE) to assess model performance.
- Employ cross-validation to validate model robustness.

Data Preprocessing:

Clean and preprocess data to handle missing values and outliers.



- NORMALIZE OR SCALE FEATURES FOR BETTER MODEL CONVERGENCE. THE HEATING, VENTILATION AND AIR CONDITIONING (HVAC) SYSTEMS ACCOUNT FOR 40%-60% OF ENERGY CONSUMPTION IN BUILDINGS.
- OPTIMAL OPERATION CONTROL FOR HVAC SYSTEMS CAN HELP IMPROVE THE ENERGY PERFORMANCE OF THE HVAC SYSTEMS DRAMATICALLY
- ACCURATE SHORT-TERM ENERGY CONSUMPTION PREDICTION IS THE ESSENTIAL PREREQUISITE FOR DEVELOPING OPTIMAL CONTROL. WITH THE DEVELOPMENT OF COMPUTER SCIENCE AND THE INCREASING ACCESSIBILITY TO LARGE AMOUNTS OF BUILDING OPERATION DATA, DATA-DRIVEN MODELS ARE WIDELY USED TO DEVELOP ENERGY CONSUMPTION PREDICTION MODELS .
-
- DATA-DRIVEN MODELS USED FOR HVAC LOAD PREDICTION ARE GENERALLY DIVIDED INTO STATISTICAL MODELS AND MACHINE LEARNING MODELS [4].
- THE STATISTICAL MODELS CAN BE CONSTRUCTED BY REGRESSION AND TIME-SERIES ANALYSIS, WHICH USUALLY DERIVE THE INFLUENCE COEFFICIENTS OF INPUT VARIABLES FOR OUTPUTS FROM THE DATA

Ensemble Approaches:

- COMBINE MULTIPLE MODELS' PREDICTIONS THROUGH TECHNIQUES LIKE STACKING OR BLENDING FOR IMPROVED ACCURACY.

Continuous Monitoring:

- CONTINUOUSLY UPDATE MODELS WITH NEW DATA TO ADAPT TO CHANGING CONSUMPTION PATTERNS.

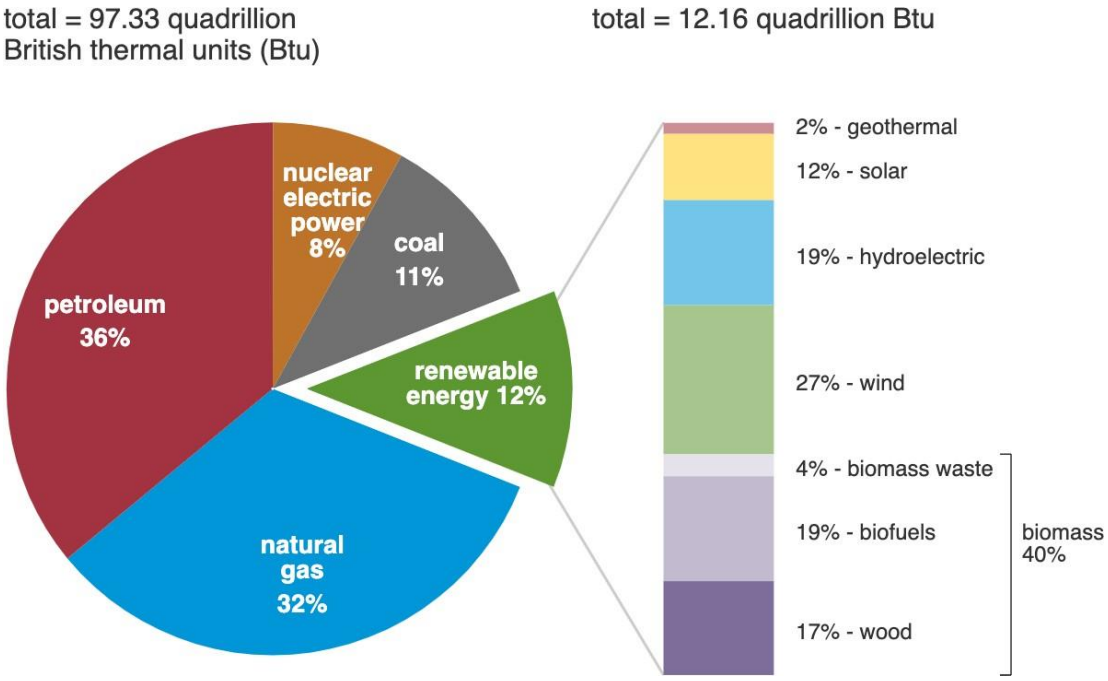
Domain Knowledge:

- INCORPORATE DOMAIN-SPECIFIC KNOWLEDGE TO REFINE MODELS FURTHER, SUCH AS UNDERSTANDING THE IMPACT OF GOVERNMENT POLICIES OR TECHNOLOGICAL ADVANCEMENTS.

THESE TECHNIQUES, WHEN APPLIED AND FINE-TUNED APPROPRIATELY, CAN PROVIDE VALUABLE INSIGHTS AND ACCURATE PREDICTIONS FOR FUTURE ENERGY CONSUMPTION PATTERNS, AIDING IN EFFECTIVE RESOURCE PLANNING AND SUSTAINABILITY EFFORTS.

- Energy consumption and industrial structure are synchronously adjusted and evolved, mutually related with each other.
- In the work, adjustment and evolution of energy consumption and industrial structure during 1978–2012 in China are verified and analyzed according to grey correlation and Granger causal theory.
- The result shows that industrial structure adjustment is the driving force of energy consumption structure evolution, giving one-way service to industrial structure evolution. Namely industrial structure adjustment is the driving force of energy consumption structure adjustment during 1978–2012 in China, thus making real energy economy unformed.

Primary Energy Consumption By Energy Source :



The objective of this article is to present the reader with a class in python that has a very intuitive and easy input to model and predict time series data using deep learning.

The data for this article can be found here:

https://www.kaggle.com/robikscube/hourly-energy-consumption

The packages that are used for deep modeling are TensorFlow and Keras.

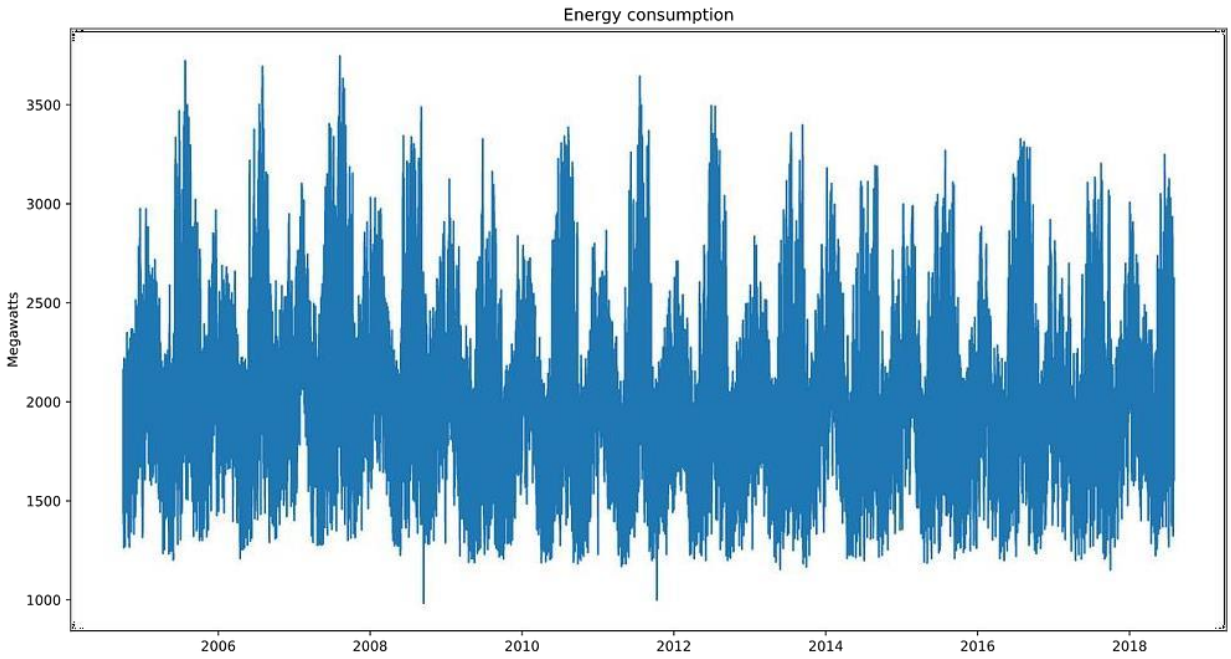
A time series is a sequence of numerical data points in successive order. These points are often measured at regular intervals (every month, every day, every hour, etc.). The data frequency used in this article is hourly and it was measured from 2004–10–01 to 2018– 08–03. The total number of raw data points is **121271**.

	Datetime	DAYTON_MW
0	2004-10-01 01:00:00	1621.0
1	2004-10-01 02:00:00	1536.0
2	2004-10-01 03:00:00	1500.0
3	2004-10-01 04:00:00	1434.0
4	2004-10-01 05:00:00	1489.0
...	...	...
121266	2018-08-02 20:00:00	2554.0
121267	2018-08-02 21:00:00	2481.0
121268	2018-08-02 22:00:00	2405.0
121269	2018-08-02 23:00:00	2250.0
121270	2018-08-03 00:00:00	2042.0

121271 rows × 2 columns

Time series example in Python

Visualization of the time series:



A line plot for the energy consumption time series

The main objective of the deep learning algorithm for a given time series is to find a function **f** such that:

$$\mathbf{Y_t = f(Y_{t-1}, Y_{t-2}, ..., Y_{t-p})}$$

In other words, we want to estimate a function that explains the current values of energy consumption based on **p** lags of the same energy consumption.

Sample program for Measuring Energy Consumption :

```
import pandas as pd

# Loading date wrangling package

from datetime import datetime

# Reading the input data

d = pd.read_csv('input/DAYTON_hourly.csv')

# Formating to datetime

d['Datetime'] = [datetime.strptime(x, '%Y-%m-%d %H:%M:%S') for x
in d['Datetime']]

# Making sure there are no duplicated data

# If there are some duplicates we average the data during those
duplicated days

d = d.groupby('Datetime', as_index=False)['DAYTON_MW'].mean()

# Sorting the values

d.sort_values('Datetime', inplace=True)
```

We then need a function that converts the time series into an X and Y matrices for the deep learning model to start learning. Let us say that we want to create a function that explains current time series values using **3** lags:

$$Y = f(Y_1, Y_2, Y_3)$$

And we have this data:

ts = [1621.0, 1536.0, 1500.0, 1434.0, 1489.0, 1620.0]

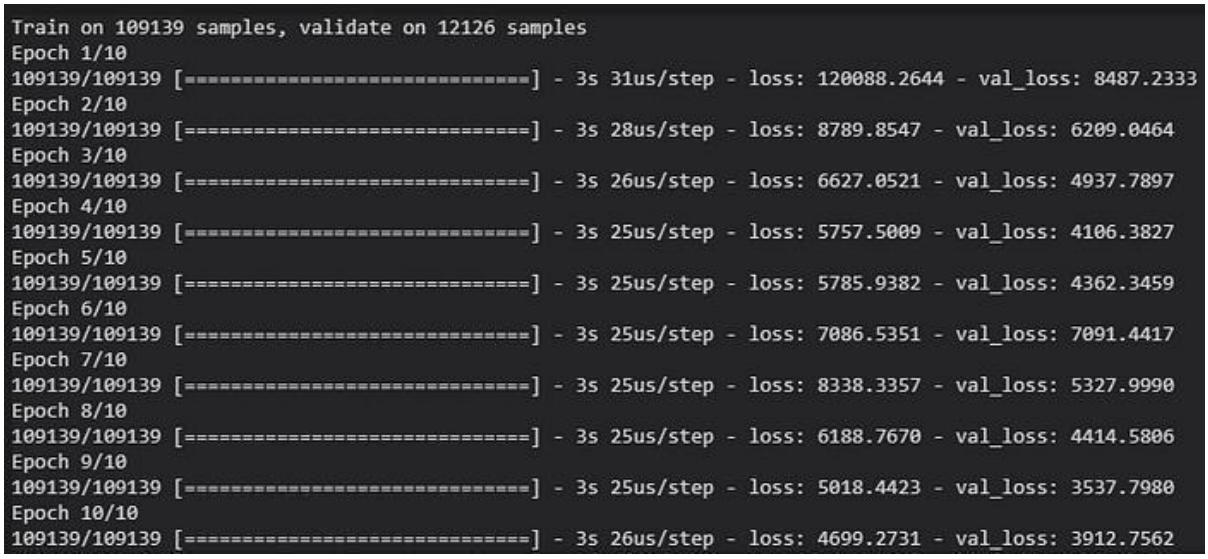
What we would like is to create two matrices:

```
X = [
[1621.0, 1536.0, 1500.0], # First three lags
[1536.0, 1500.0, 1434.0], # Second three lags
[1500.0, 1434.0, 1489.0], # Third three lags
]Y = [1434.0, 1489.0, 1620.0]
```

Fitting the model:

```
# Fitting the model
model = deep_learner.LSTModel()
```

After the above command, you can witness the fan-favorite training screen:



Training the model with more lags (hence, a larger X matrix) increases the training time:

```
deep_learner = DeepModelTS(
data = d,
Y_var = 'DAYTON_MW',
lag = 24, # 24 past hours are used
LSTM_layer_depth = 50,
epochs = 10,
batch_size = 256,
train_test_split = 0.15
)model = deep_learner.LSTModel()
```

```
Train on 109122 samples, validate on 12125 samples
Epoch 1/10
109122/109122 [==============================] - 13s 119us/step - loss: 198459.0185 - val_loss: 42519.4667
Epoch 2/10
109122/109122 [==============================] - 13s 116us/step - loss: 55858.1046 - val_loss: 48171.6780
Epoch 3/10
109122/109122 [==============================] - 13s 122us/step - loss: 56637.3061 - val_loss: 59201.6064
Epoch 4/10
109122/109122 [==============================] - 13s 123us/step - loss: 59519.4859 - val_loss: 50751.3374
Epoch 5/10
109122/109122 [==============================] - 13s 117us/step - loss: 30777.7760 - val_loss: 21393.2249
Epoch 6/10
109122/109122 [==============================] - 13s 116us/step - loss: 24113.7349 - val_loss: 19821.5885
Epoch 7/10
109122/109122 [==============================] - 13s 115us/step - loss: 21828.8772 - val_loss: 20949.0823
Epoch 8/10
109122/109122 [==============================] - 12s 114us/step - loss: 20531.1940 - val_loss: 16574.5223
Epoch 9/10
109122/109122 [==============================] - 13s 116us/step - loss: 18882.8906 - val_loss: 14898.6707
Epoch 10/10
109122/109122 [==============================] - 13s 117us/step - loss: 17565.6644 - val_loss: 15311.7685
```

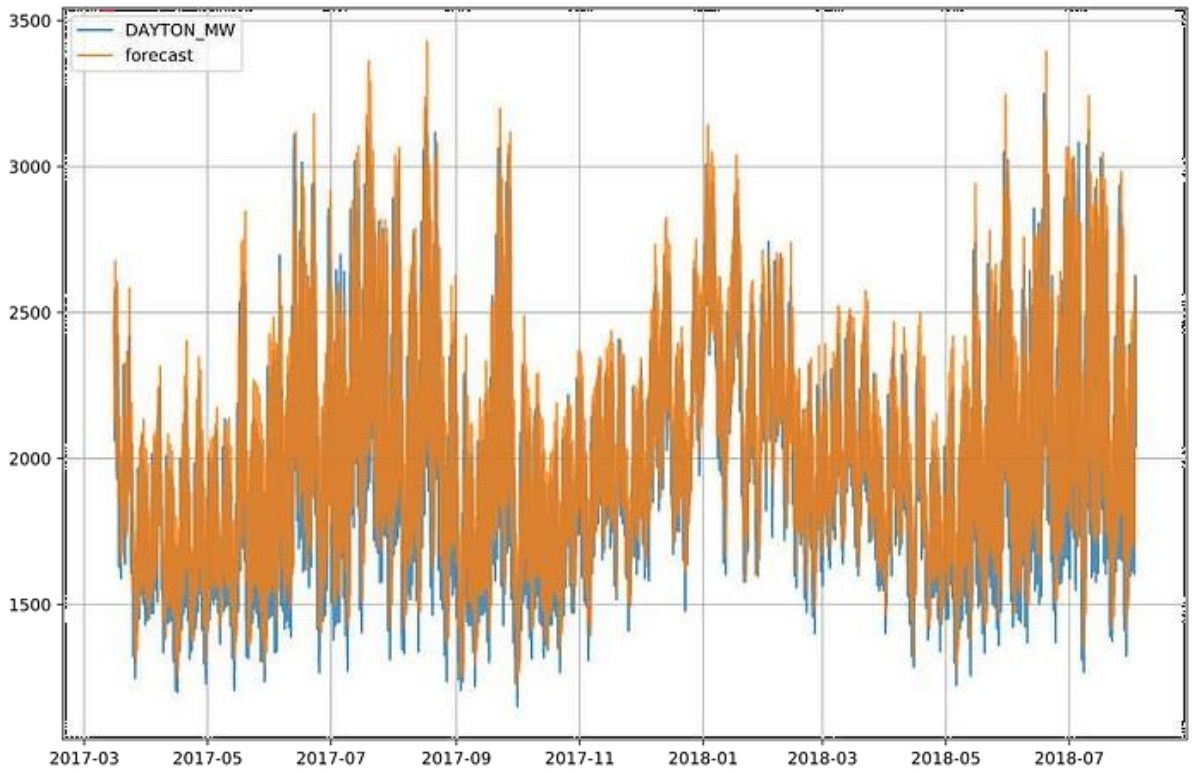
Now that we have a created model we can start forecasting. The formula for the forecasts with a model trained with **p** lags:

```
# Defining the lag that we used for training of the model
lag_model = 24# Getting the last period
ts = d['DAYTON_MW'].tail(lag_model).values.tolist()# Creating
the X matrix for the model
X, _ = deep_learner.create_X_Y(ts, lag=lag_model)# Getting the
forecast
yhat = model.predict(X)
```

If the data was split into training and test sets then the **deep_learner.predict()** method will predict the points which are in the test set to see how our model performs out of sample.

```
yhat = deep_learner.predict()# Constructing the forecast
dataframe
fc = d.tail(len(yhat)).copy()
fc.reset_index(inplace=True)
fc['forecast'] = yhat# Ploting the forecasts
plt.figure(figsize=(12, 8))
for dtype in ['DAYTON_MW', 'forecast']:  plt.plot(
    'Datetime',
    dtype,
    data=fc,
    label=dtype,
```

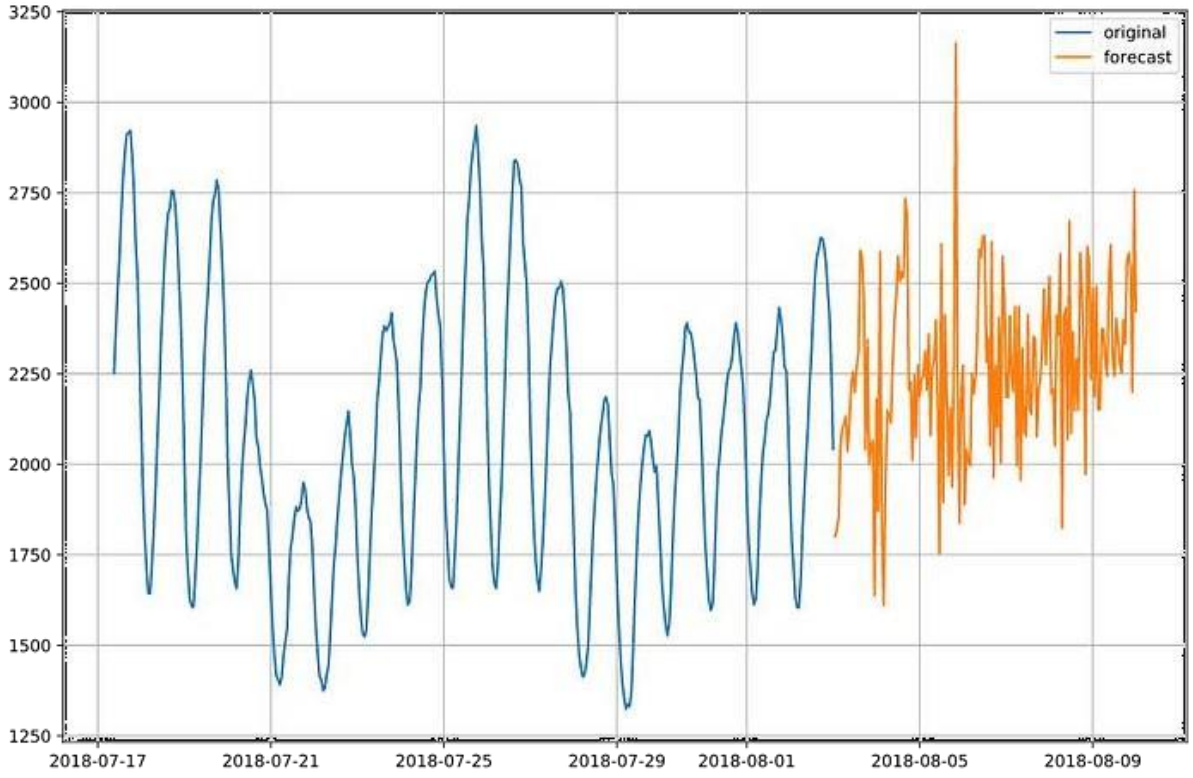
```
    alpha=0.8
  )plt.legend()
plt.grid()
plt.show()
```



We usually want to forecast ahead of the last original time series data. The class **DeepModelTS** has the method **predict_n_ahead(n_ahead)** which forecasts **n_ahead** time steps.

```
# Creating the model using full data and forecasting n steps
aheaddeep_learner = DeepModelTS(
data=d,
Y_var='DAYTON_MW',
lag=48,
LSTM_layer_depth=64,
epochs=10,
train_test_split=0
)# Fitting the model
deep_learner.LSTModel()# Forecasting n steps ahead
n_ahead = 168yhat = deep_learner.predict_n_ahead(n_ahead)
yhat = [y[0][0] for y in yhat]
```

The above code forecasts one week's worth of steps ahead (168 hours). Comparing it with the previous 400 hours:


original
forecast
3250
3000
2750
2500
2250
2000
1750
1500
1250
2018-07-17
2018-07-21
2018-07-25
2018-07-29
2018-08-01
2018-08-05
2018-08-09

- Experiments relating to energy measurement could be at various levels: the hardware level; energy efficiency directive level (Simunic, et al. 2000); operating system (Sagahyroon, 2006); software application or data and user levels (Ravi, et al. 2008). Energy conservation is made possible through the use of different techniques which estimate or forecast energy consumption at the device and application level (Krintz, et al. 2004).
- The goal of green computing technology is to reduce carbon emission, maximize performance and prolong the lifespan of the computing resources

1.Literature Review

- The Smart2020 report (The Climate Group and GeSI, 2008) predicts an increasing trend of BAU CO2 emissions for the ICT industry. The emissions growth rate for three ICT categories (end-user devices, telecommunication and networks, and data centers) is expected to decrease from 6.1% 3.8%. By 2020, the ICT industry's footprint is expected to rise to 1.3 GtCO2e (equivalent to 2.3% of global emissions by 2020).
- The PC (e.g. desktops, laptops, etc.) footprint (due to its embodied and usage emissions) is the highest (60%) followed by printers (18%), peripherals (13%), smartphones (10%), and tablets (1%). It is estimated that the footprint of end-user devices will grow at 2.3 percent per year to reach 0.67 GtCO2e in 2020 and thus, energy efficiency improvements in these devices and their proper usage are essential for reducing their overall footprint.

Energy Consumption of Media Players

- Modern technologies incorporate a number of power management features to reduce power waste. Dynamic Voltage and Frequency Scaling (DVFS) can enable the CPU speed to be dynamically varied based on the workload which leads to a reduced power consumption during periods of low utilization (Liu, et al., 2008).
- The energy-aware dynamic voltage scaling technique has been used to reduce energy consumption in portable media players (Yang & Song, 2009). This scheme showed a relationship between frame size and decoding time. These two cited work merely discuss how energy consumption can be reduced using various techniques, but have not measured the actual amount of energy being consumed by the

application. However, the energy consumption of Windows Media Player has been measured using the EEcoMark v2 tool (EecoMark, 2011) but the empirical details of the measurement have not been explicitly discussed. Media playback application power consumption has been analysed by Sabharwal (2011) using windows event tracing.

- Event tracing does not seem to be an appropriate method for measuring energy consumption because the process itself may have impact on the results. A comparative analysis of energy consumption of media players has been conducted by Techradar (2010).
- The energy consumption is monitored by playing a DVD on Windows Media Player (WMP) and VLC Media Player.
- Their research results show that the VLC Media Player is more energy efficient than Windows Media Player. However, the cited work has not mentioned which tool has been used for measurement and additionally, the experiment procedures have not been explitcitly discussed.

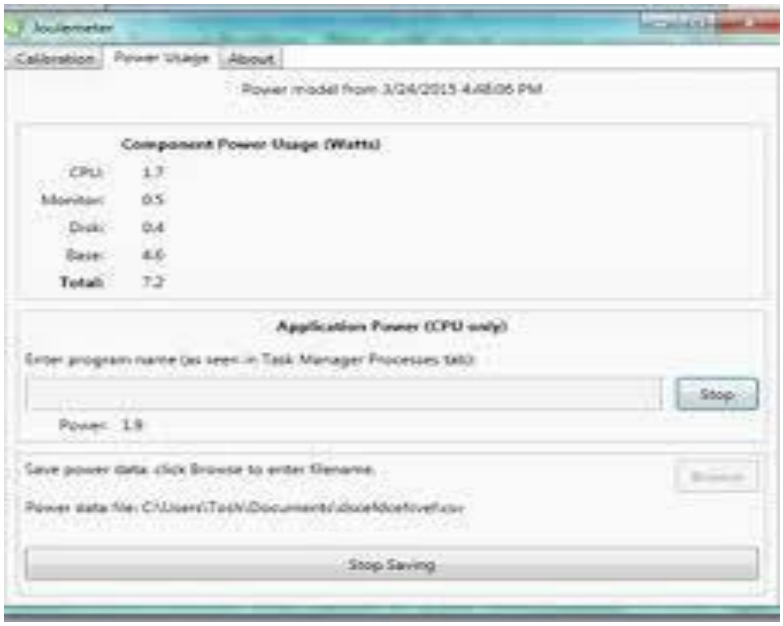
Tools and measure

- Power models are used to calculate the energy consumption of hardware and software. Kansal and Zhao (2008) use a generic automated tool to profile the energy usage of various resources components used by an application.
- This method is either too generic or coarse-grained and it is platform dependent (Seo, et al., 2007). The model proposed by Lewis and colleagues (2012) is an integrated model for the calculation of a system's energy consumption.
- More promising approaches are software energy measurement using energy application profiler (Noureddine, et al., 2013). In their contribution, Varrol and Heiser (2010) use Openmoko Neo Freerunner to decompose the energy consumption of each resource of a system. PowerAPI is an Application Programming Interface (API) used for monitoring the real time energy consumption of applications at the granularity of system process (Bourdon, et al., 2013).
- PowerAPI can also be used to estimate the energy consumption of a running process for hardware resources e.g. CPU or for hard disk or for both and many more other resources (Noureddine, et al., 2013). Energy consumption estimation in PowerAPI distinguishes the energy consumption for hardware resources and software blocks of codes.

Experimental Procedures

Experiment Set 1: To investigate the energy consumption of several web browser applications in Windows (the sample interface is shown in Figure 1)
Experimental Steps

- Create a csv file for saving the real time power consumption data via Joulemeter (by clicking on the browse button);
- Click on the start saving button;
- Click on the start button to run the application in Google Chrome 1.3.27 (i.e a youtube video3);
- Click on the stop saving button to end the application; v. Repeat the above steps for 9 times;
- Repeat all the above steps for each of the following web browser: Internet Explorer 9;



LOADING DATASET

Data loading in earthquake prediction is a crucial step in the process. It involves gathering and organizing relevant information to train and test predictive models

DOWNLOADING DATASET:

https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption

IMPORTING LIVRARIES

```
# Creating the model using full data and forecasting n steps aheaddeep_learner =
DeepModelTS(
data=d,
Y_var='DAYTON_MW',
lag=48,
LSTM_layer_depth=64,
epochs=10,
train_test_split=0
)# Fitting the model
deep_learner.LSTModel()# Forecasting n steps ahead
n_ahead = 168yhat = deep_learner.predict_n_ahead(n_ahead)
yhat = [y[0][0] for y in yhat]
```

Loading a dataset

- This section will discuss the results of the data analysis for the three sets of experiments discussed above. The Joulemeter monitored raw data is for the time stamp (in ms), power consumption (in Watts) for each component:

- CPU, monitor, disk, base and the application. The formula used to calculate the energy consumption by each component is: Energy (J) = Power (W) x Time (s). The results of the calculation for all the experiments runs are shown in Table 2. Note that the data is cleansed so as to omit records with application power consumption = 0W. If the number of remaining records > 50% of the raw data then the cleansed readings of the csv file will be included in the data analysis. However, if it is otherwise, then the experiment is considered an error (see Table 2). 4.1 Web Browser Applications for Windows Note: * - proportion of the remaining cleansed data > 50% of the raw data and ** is for otherwise and thus considered an error. White coloured records – results of experiments for Day 1; Blue coloured records – results of experiments for Day 2. Table 2: The hardware and application energy consumption for running several web browser

applications Table 3 depicts the aggregated data for all the experiments conducted for each web browser: Google Chrome, Internet Explorer, Mozilla Firefox, and Safari. However, in order to provide a fair comparison among the web browsers, the time for running the application will have to be set to 1s (i.e. t = 1s) Consequently, the corresponding energy consumption for each component will have to be normalised for t = 1s (see Table 4). Based on the results shown in Table 4, it seems that Internet Explorer 9 consumes the least energy on laptops, followed by Mozilla Firefox and Safari while Google Chrome seems to be the highest energy consumer.

- These results are consistent with experiments conducted by the Center for Sustainable Energy Systems at Fraunhofer USA 4 , which compare the energy consumption of Internet Explorer 10, Mozilla Firefox and Google Chrome on laptops and desktops. Their results reveal that Google Chrome consumes the highest amount of energy followed by Mozilla Firefox. The conclusion drawn by them is that Internet

Program

```
import numpy as np

import pandas as pd



# Deep learning:

from keras.models import Sequential

from keras.layers import LSTM, Dense



class DeepModelTS():
```

```
"""
A class to create a deep time series model
"""
def __init__(
    self,
    data: pd.DataFrame,
    Y_var: str,
    lag: int,
    LSTM_layer_depth: int,
    epochs=10,
    batch_size=256,
    train_test_split=0
):



    self.data = data
    self.Y_var = Y_var
    self.lag = lag
     self.LSTM_layer_depth = LSTM_layer_depth
    self.batch_size = batch_size
    self.epochs = epochs
     self.train_test_split = train_test_split



@staticmethod
def create_X_Y(ts: list, lag: int) -> tuple:
    """
```

```
        A method to create X and Y matrix from a time series list for the
training of

        deep learning models

        """

        X, Y = [], []



        if len(ts) - lag <= 0:

            X.append(ts)

        else:

            for i in range(len(ts) - lag):

                Y.append(ts[i + lag])

                X.append(ts[i:(i + lag)])



        X, Y = np.array(X), np.array(Y)



         # Reshaping the X array to an LSTM input shape

         X = np.reshape(X, (X.shape[0], X.shape[1], 1))



        return X, Y



    def create_data_for_NN(

        self,

        use_last_n=None

        ):
```

```
"""
A method to create data for the neural network model
"""
# Extracting the main variable we want to model/forecast
y = self.data[self.Y_var].tolist()


# Subseting the time series if needed
if use_last_n is not None:
    y = y[-use_last_n:]


# The X matrix will hold the lags of Y
X, Y = self.create_X_Y(y, self.lag)


# Creating training and test sets
X_train = X
X_test = []


Y_train = Y
Y_test = []


if self.train_test_split > 0:
    index = round(len(X) * self.train_test_split)
    X_train = X[:(len(X) - index)]
```

```
            X_test = X[-index:]


            Y_train = Y[:(len(X) - index)]
            Y_test = Y[-index:]



        return X_train, X_test, Y_train, Y_test



    def LSTModel(self):
        """
        A method to fit the LSTM model
        """
        # Getting the data
        X_train, X_test, Y_train, Y_test = self.create_data_for_NN()



        # Defining the model
        model = Sequential()
        model.add(LSTM(self.LSTM_layer_depth, activation='relu',
input_shape=(self.lag, 1)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')



        # Defining the model parameter dict
        keras_dict = {
            'x': X_train,
            'y': Y_train,
```

```
        'batch_size': self.batch_size,
        'epochs': self.epochs,
        'shuffle': False
    }


    if self.train_test_split > 0:
        keras_dict.update({
            'validation_data': (X_test, Y_test)
        })


    # Fitting the model
    model.fit(
        **keras_dict
    )


    # Saving the model to the class
    self.model = model


    return model


def predict(self) -> list:
    """
    A method to predict using the test data used in creating the class
```

```
    """
    yhat = []

    if(self.train_test_split > 0):

        # Getting the last n time series
        _, X_test, _, _ = self.create_data_for_NN()

        # Making the prediction list
        yhat = [y[0] for y in self.model.predict(X_test)]

    return yhat

def predict_n_ahead(self, n_ahead: int):
    """
    A method to predict n time steps ahead
    """
    X, _, _, _ = self.create_data_for_NN(use_last_n=self.lag)

    # Making the prediction list
    yhat = []
```

```
for _ in range(n_ahead):
    # Making the prediction
    fc = self.model.predict(X)
    yhat.append(fc)

    # Creating a new input matrix for forecasting
    X = np.append(X, fc)

    # Ommiting the first variable
    X = np.delete(X, 0)

    # Reshaping for the next iteration
    X = np.reshape(X, (1, len(X), 1))

return
```

- To obtain a benchmark for comparison against the performance of the recurrent network models, two models were developed. The first is the last observation carry forward (LOCF).
- It is a naïve model in which the one-step ahead prediction is the last observed value in the sequence. The MAE calculated for the LOCF model on the validation split using a simple 2:1 split is 3.983.
- Despite being extremely simple, the LOCF model otherwise also known as simple persistence forecasting, is nonetheless a credible benchmark as research in forecasting has shown. Outperforming the LOCF model

- I can guide you through creating some common types of visualizations and provide examples of how to do it using tools like Python with libraries such as Matplotlib or Seaborn.

- Here are a few popular visualizations you might consider:

1. Time Series Line Plot:

   - This plot shows energy consumption over time. You can create a line plot with time on the x-axis and energy consumption on the y-axis.

2. Bar Chart:

   - A bar chart can show the energy consumption for different categories or time periods, such as monthly or yearly consumption.

3. Histogram:

   - Histograms can help you understand the distribution of energy consumption values.

4. Box Plot:

   - Box plots are useful for visualizing the distribution of energy consumption and identifying outliers.

5. Heatmap:

- A heatmap can show energy consumption patterns over time, helping you identify high and low consumption periods.

6. Scatter Plot:

- If you have multiple variables, you can create scatter plots to see if there are correlations between energy consumption and other factors.

Here's a simple example using Python and Matplotlib to create a time series line plot:

Python Program :

```
import matplotlib.pyplot as plt

import pandas as pd

# Load your energy consumption data into a DataFrame

# Replace 'data.csv' with your actual data file.

data = pd.read_csv('data.csv')

# Assuming your data has a 'timestamp' and 'consumption' column.
```

Convert the 'timestamp' column to a datetime object if it's not already.

data['timestamp'] = pd.to_datetime(data['timestamp'])

Create the line plot

plt.figure(figsize=(10, 6))

plt.plot(data['timestamp'], data['consumption'])

plt.xlabel('Time')

plt.ylabel('Energy Consumption')

plt.title('Energy Consumption Over Time')

plt.grid(True)

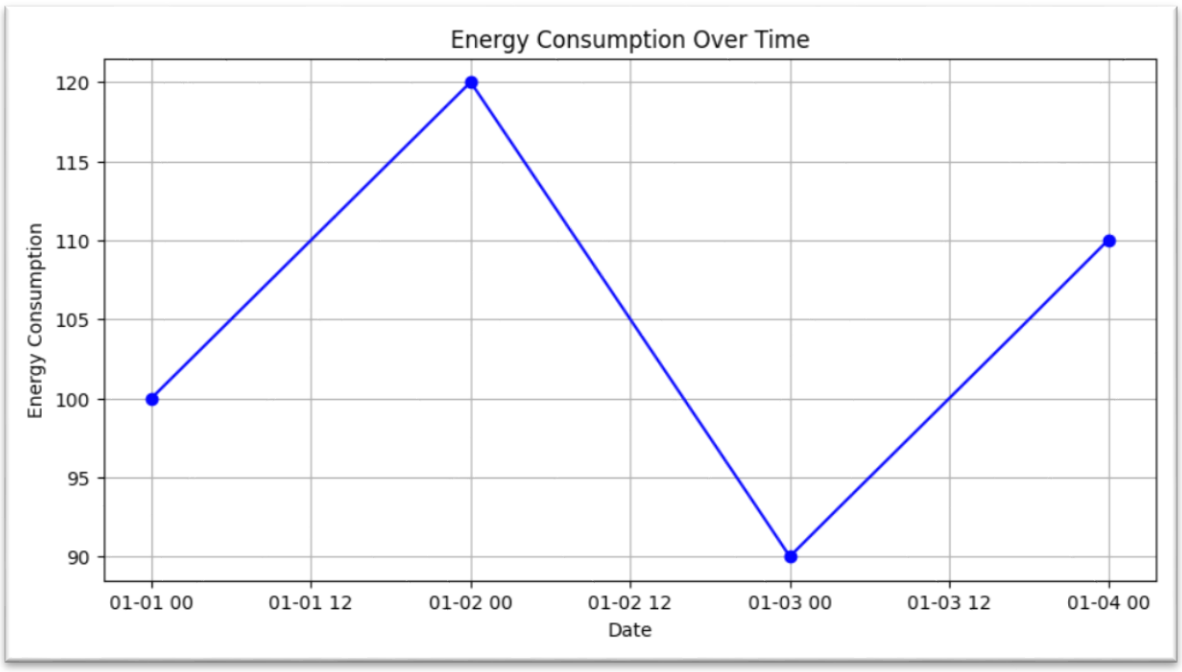
plt.show()

EXAMPLE PROGRAM :

```
import matplotlib.pyplot as plt
import pandas as pd

# Sample energy consumption data (replace with your own dataset)
data = {
    'Date': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04'],
    'Energy Consumption': [100, 120, 90, 110]
}
```

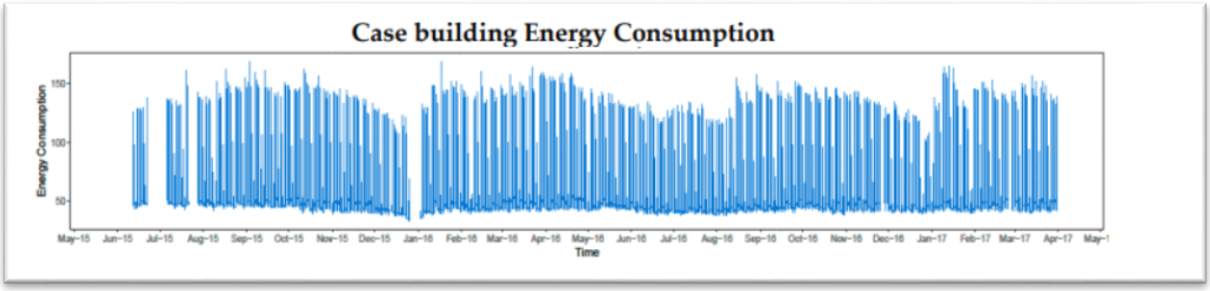
```
df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])

plt.figure(figsize=(10, 5))
plt.plot(df['Date'], df['Energy Consumption'], marker='o', linestyle='-
', color='b')
plt.title('Energy Consumption Over Time')
plt.xlabel('Date')
plt.ylabel('Energy Consumption')
plt.grid(True)
plt.show()
```

OUTPUT :



Case Building and Case Dataset :

- The data comprises the electricity consumption and cooling thermal energy of a mixed function institutional building in a university campus in Singapore. Each data is measured at a half-hourly frequency from June 2015 to March 2017. As seen in the plots of the data (Figure 1), there is missing data in both datasets.



The energy consumption data exhibits weekly cycles (Figure 2). The plot shows the energy consumption for a typical week. Energy consumption peaks during office hours and reduces to a minimum in the hours of the night and early morning. The energy consumption is also lower on Saturdays and lowest on Sundays. The plot also shows a clear correlation between the energy consumption and cooling capacity in use.

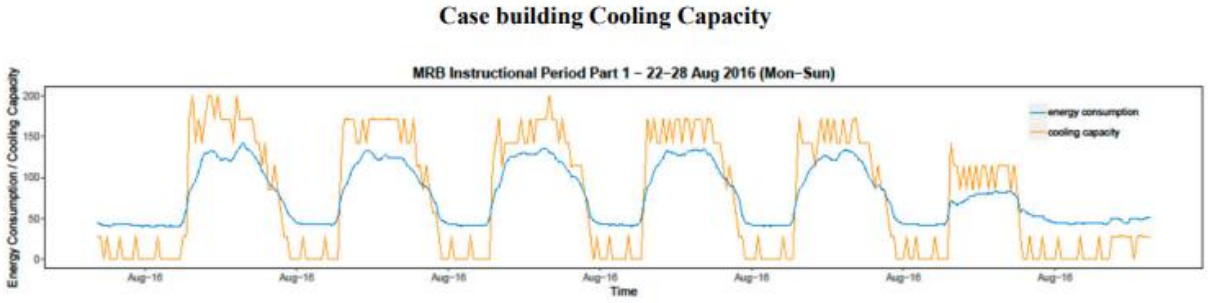


Figure 2. Weekly cycles in the data.

Data Transformation :

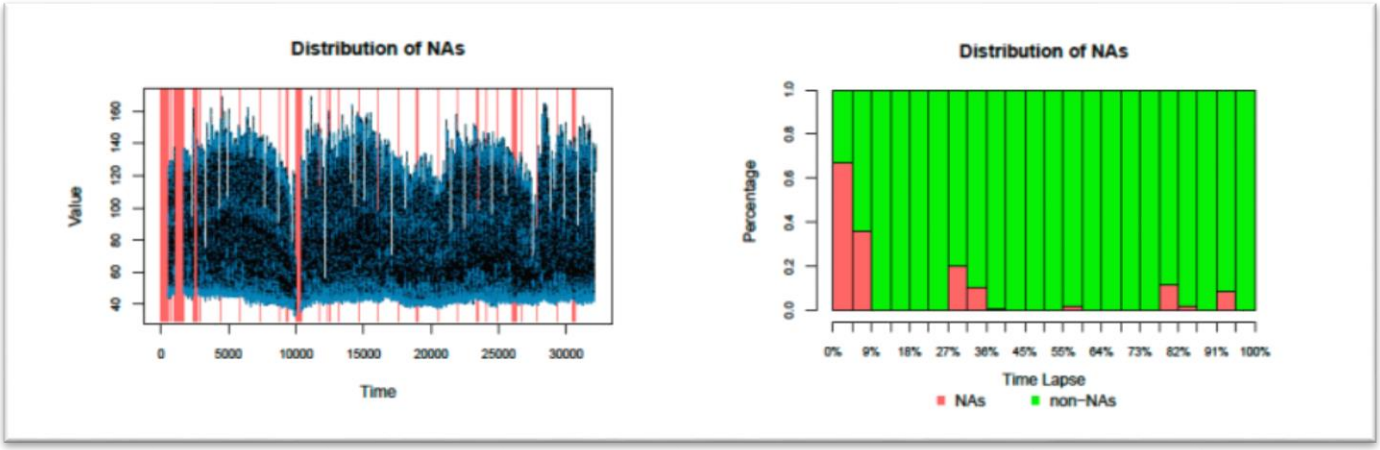
- Data transformation in energy consumption refers to the process of modifying or converting raw energy consumption data into a more useful and informative format. This transformation is often necessary to make the data suitable for analysis, visualization, or modeling.

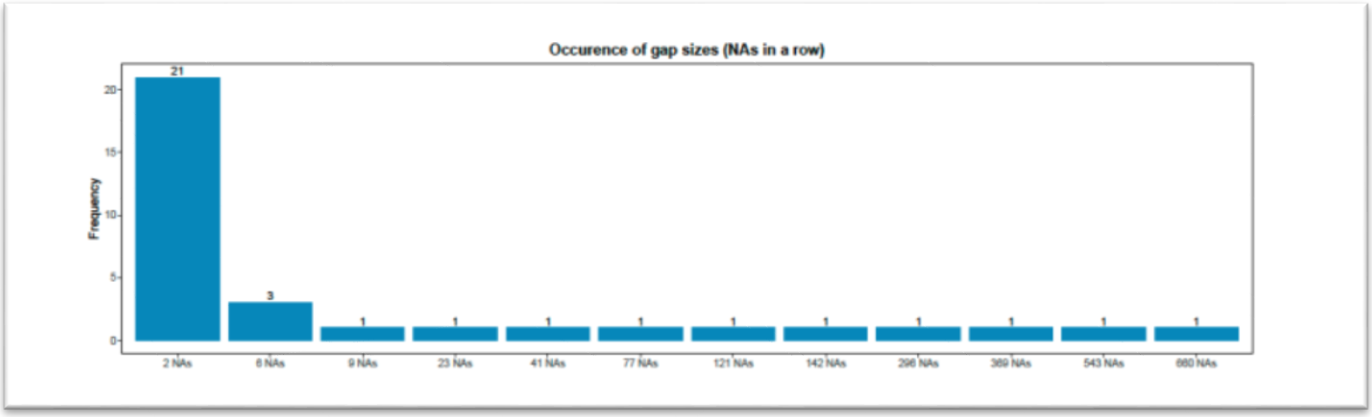
```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

```
# Sample energy consumption data in a CSV file
data_file = "energy_consumption_data.csv"
# Load the data into a Pandas DataFrame
df = pd.read_csv(data_file)
# View the first few rows of the data
print("Original Data:")
print(df.head())
# Data Transformation Steps
# 1. Aggregation - Convert hourly data to daily data by summing consumption
df['Date'] = pd.to_datetime(df['Timestamp'])
daily_data = df.resample('D', on='Date').sum()
# 2. Normalization - Scale the data using min-max scaling
daily_data['Consumption (Normalized)'] = (daily_data['Consumption'] - daily_data['Consumption'].min()) / (daily_data['Consumption'].max() - daily_data['Consumption'].min())
# 3. Data Visualization - Plot the original and normalized data
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(daily_data['Date'], daily_data['Consumption'])
plt.title("Original Daily Consumption")
plt.subplot(2, 1, 2)
plt.plot(daily_data['Date'], daily_data['Consumption (Normalized)'])
plt.title("Normalized Daily Consumption")
plt.tight_layout()
```

```
plt.show()
```

```
# 4. Seasonal Decomposition - Decompose the data into seasonal, trend, and residual components

from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(daily_data['Consumption'], model='additive', freq=365)  # Assumes a yearly seasonality

# 5. Data Visualization - Plot the decomposition components

plt.figure(figsize=(12, 6))

result.plot()

plt.show()

# 6. Data Categorization - Categorize data into different sectors

df['Sector'] = pd.cut(df['Consumption'], bins=[0, 500, 1000, 1500], labels=['Low', 'Medium', 'High'])

# View the first few rows of the categorized data

print("Categorized Data:")

print(df.head())
```





Conclusion :

- While this work has raised questions regarding building energy data forecasting, it is computationally intensive to test on each model, especially when more data will be used to train the model.
- One future work is to look at the application of transfer learning in the building energy data forecasting to reduce the computational time when applying the same method to more buildings or meters.
- Another direction is to de-trend, or de-seasonalize data according to building function and occupancy patterns, which could increase the accuracy and reduce additional computational time.